**Video:**

[**https://drive.google.com/drive/folders/1h39IYJA0lg1mcB\_e9bG1pHTJVXvPQOow?usp=sharing**](https://drive.google.com/drive/folders/1h39IYJA0lg1mcB_e9bG1pHTJVXvPQOow?usp=sharing)

**Instrucciones:**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Código solución del proyecto:**

import sys

import numpy as np

# Simula el ataque de un grupo a otro

def atacar(matriz, grupos):

    # Creamos un vector para registrar el número original de cada grupo

    grupo\_numeros = np.arange(len(grupos)) + 1

    while np.count\_nonzero(grupos) > 1: # Mientras exista más de un grupo vivo

        # Seleccionamos aleatoriamente el grupo atacante de los grupos que aún tienen guerreros y tienen posibles víctimas

        posibles\_atacantes = np.where((grupos > 0) & (np.sum(matriz > 0, axis=1) > 0))[0]

        if len(posibles\_atacantes) == 0:  # Si no hay posibles atacantes, se sale del bucle

            break

        atacante = np.random.choice(posibles\_atacantes)

        # Seleccionamos las posibles víctimas del ataque (aquellos grupos a los que el grupo atacante puede atacar)

        posibles\_victimas = np.where(matriz[atacante] > 0)[0]

        # Seleccionamos aleatoriamente la víctima del ataque en base a las probabilidades de la matriz estocástica

        victima = np.random.choice(posibles\_victimas, p=matriz[atacante][posibles\_victimas])

        # Reducimos el número de guerreros del grupo víctima

        grupos[victima] -= 1

        print(f"Grupo {grupo\_numeros[atacante]} ataco al Grupo {grupo\_numeros[victima]}!")

        # Si el grupo víctima ha sido aniquilado, reconfiguramos la matriz estocástica y el vector de grupos

        if grupos[victima] == 0:

            print(f"Grupo {grupo\_numeros[victima]} ha sido aniquilado!\n")

            matriz = reconfigurar\_matriz(matriz, victima)

            grupos = np.delete(grupos, victima)  # Elimina la entrada correspondiente al grupo aniquilado

            grupo\_numeros = np.delete(grupo\_numeros, victima)  # Actualiza el vector de números de grupo

            print("Reconfigurando matriz estocastica\n")

            imprimir\_matriz(matriz)

        print("Numero de guerreros por cada grupo")

        imprimir\_grupos(grupos)

    print("================[Fin de batalla]===============\n")

    ganador = np.where(grupos > 0)[0]

    if ganador.size > 0:

        print("El ganador es el grupo", grupo\_numeros[ganador[0]], "\n")

# Reconfigura la matriz estocástica cuando un grupo es aniquilado

def reconfigurar\_matriz(matriz, indice):

    matriz = np.delete(matriz, indice, 0) # Eliminamos la fila correspondiente al grupo aniquilado

    matriz = np.delete(matriz, indice, 1) # Eliminamos la columna correspondiente al grupo aniquilado

    # Normalizamos las filas de la matriz para que sigan siendo estocásticas

    for i in range(len(matriz)):

        suma\_de\_fila = np.sum(matriz[i])

        # Si la suma total de la fila es diferente de cero, dividimos cada elemento por la suma total

        if suma\_de\_fila != 0:

            matriz[i] /= suma\_de\_fila

    return matriz

# Imprimir la matriz

def imprimir\_matriz(matriz):

    for i in range(len(matriz)):

        for j in range(len(matriz[i])):

            print(round(matriz[i][j], 2), end="\t")

        print("\n")

    print("================[Matriz actualizada]=================\n")

# Imprimir los grupos

def imprimir\_grupos(grupos):

    for i in range(len(grupos)):

        print(f"Grupo {i + 1}: {grupos[i]}")

    print("================[Grupos actualizados]=================\n")

# Genera la matriz estocástica de forma aleatoria

def crear\_matriz\_aleatoria(n):

    matriz = np.random.random((n, n)) # Crea la matriz con numeros aleatorios

    np.fill\_diagonal(matriz, 0) # Previene el ataca entre grupos

    # Normalizamos las filas para que sean estocásticas

    matriz /= matriz.sum(axis=1, keepdims=True)

    return matriz

# Genera la cantidad de guerreros por grupo de forma aleatoria

def generar\_guerreros(n):

    guerreros = np.random.randint(5, 15, n) # Guerreros por grupo entre 5 y 15

    return guerreros

# Función principal del programa

def main():

    # Número de grupos

    n = int(input("Ingresa a cantidad de grupos 🦸: "))

    # Generamos los grupos y la matriz estocástica de forma aleatoria

    grupos = generar\_guerreros(n)

    matriz = crear\_matriz\_aleatoria(n)

    # Permite enviar a un texto la salida por consola

    sys.stdout = open('output.txt', 'w')

    print("========= Matriz inicial =========")

    imprimir\_matriz(matriz)

    print("========= Numero de guerreros inicial  =========")

    imprimir\_grupos(grupos)

    # Esta función permite simular la batalla. ⚔️

    atacar(matriz, grupos)

# Condicional de arranque

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Descripción:**

Este proyecto se divide en 6 funciones de operación y 1 de arranque. En total 7.

La función main llama en orden secuencial la ejecución del programa.

Función atacar: Simula el ataque de un grupo a otro.

Función reconfigurar\_matriz: Reconfigura la matriz estocástica cuando un grupo es aniquilado.

Función imprimir\_matriz: Imprime la matriz.

Función imprimir\_grupos: Imprime los grupos.

Función crear\_matriz\_aleatoria: Genera la matriz estocástica de forma aleatoria.

Función generar\_guerreros: Genera la cantidad de guerreros por grupo de forma aleatoria

**Conclusión:**

En este proyecto, me enfrenté a diversos retos trabajando con simulaciones y matrices estocásticas, tales como errores de índice, bucles infinitos y asegurar la aleatoriedad del proceso. Aprendí que estos desafíos se pueden superar con un enfoque metódico y persistente, corrigiendo los problemas uno por uno hasta obtener un código que funcioné como se desea.

Me di cuenta de que, en la programación, los errores son parte del camino, pero con paciencia y perseverancia, se pueden manejar de manera efectiva. Entendí la importancia de validar los resultados y corregir los bugs a medida que aparecen. Al final, logré desarrollar una simulación de batalla que funcionó correctamente, lo que demuestra que, con esfuerzo y dedicación, es posible crear sistemas complejos y eficientes. Sin duda, estos aprendizajes me serán de gran utilidad en mi futuro como ingeniero.